Anotações:

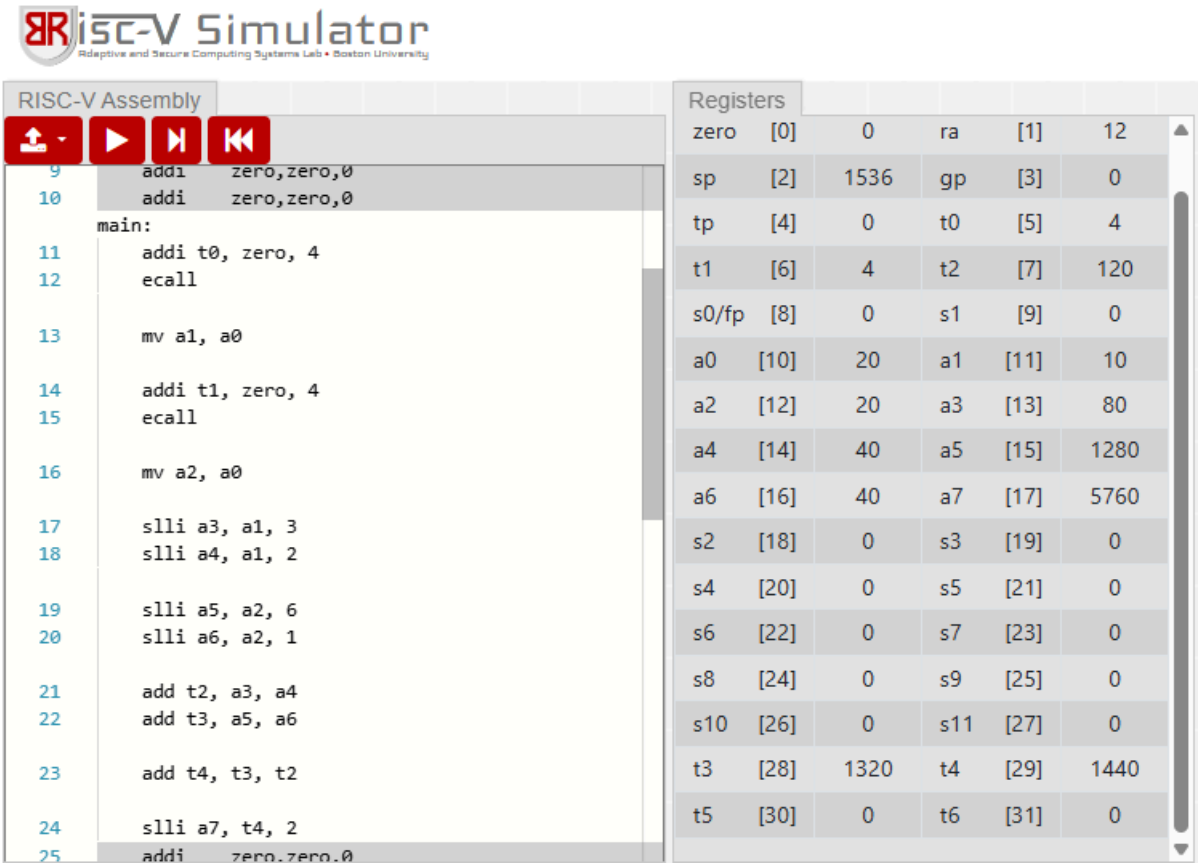
SLL = Shift Left Logical
SRL = Shift Right Logical

Atividade 1

Faça um programa em assembly que que leia x e y do teclado, e realize a equação (utilizando deslocamentos):

`z=((12*x)+(66*y))*4;`

Compare o comportamento com o uso das instruções **mul** e **div** (quando disponíveis).



Responda:

- Qual instrução é mais simples de executar no nível de hardware?

  Os deslocadores de bits: SRL e SLL.

- O que acontece se o número for negativo e você usar `srl` (deslocamento lógico à direita)?

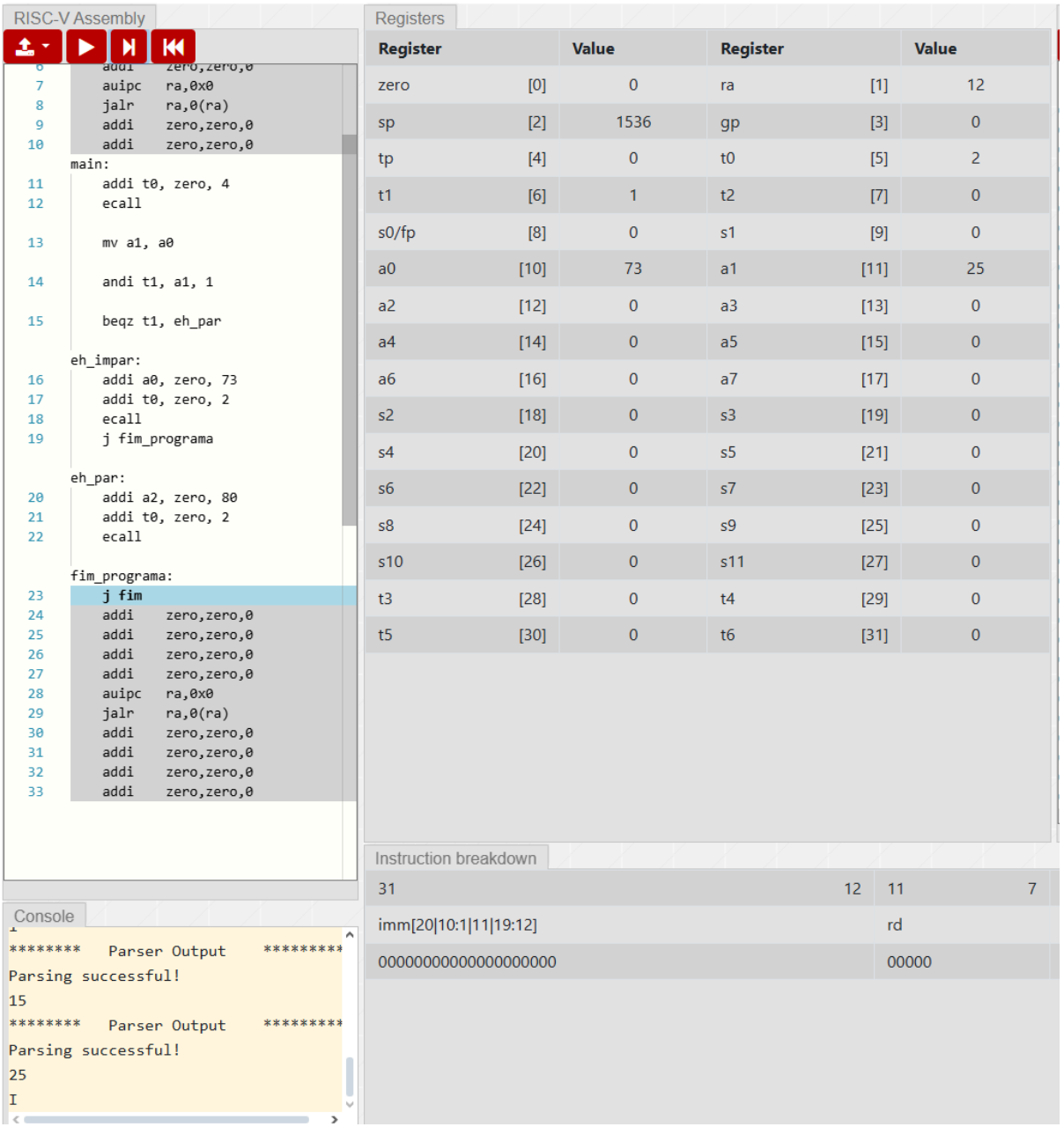
  Ele vira um número positivo, e muda o valor. Por exemplo, -8 é 11111000, ao usar o srl ele vira 01111100.

- Qual instrução deve ser usada para preservar o sinal?

  sar (Shift Arithmetic Right).

Atividade 2

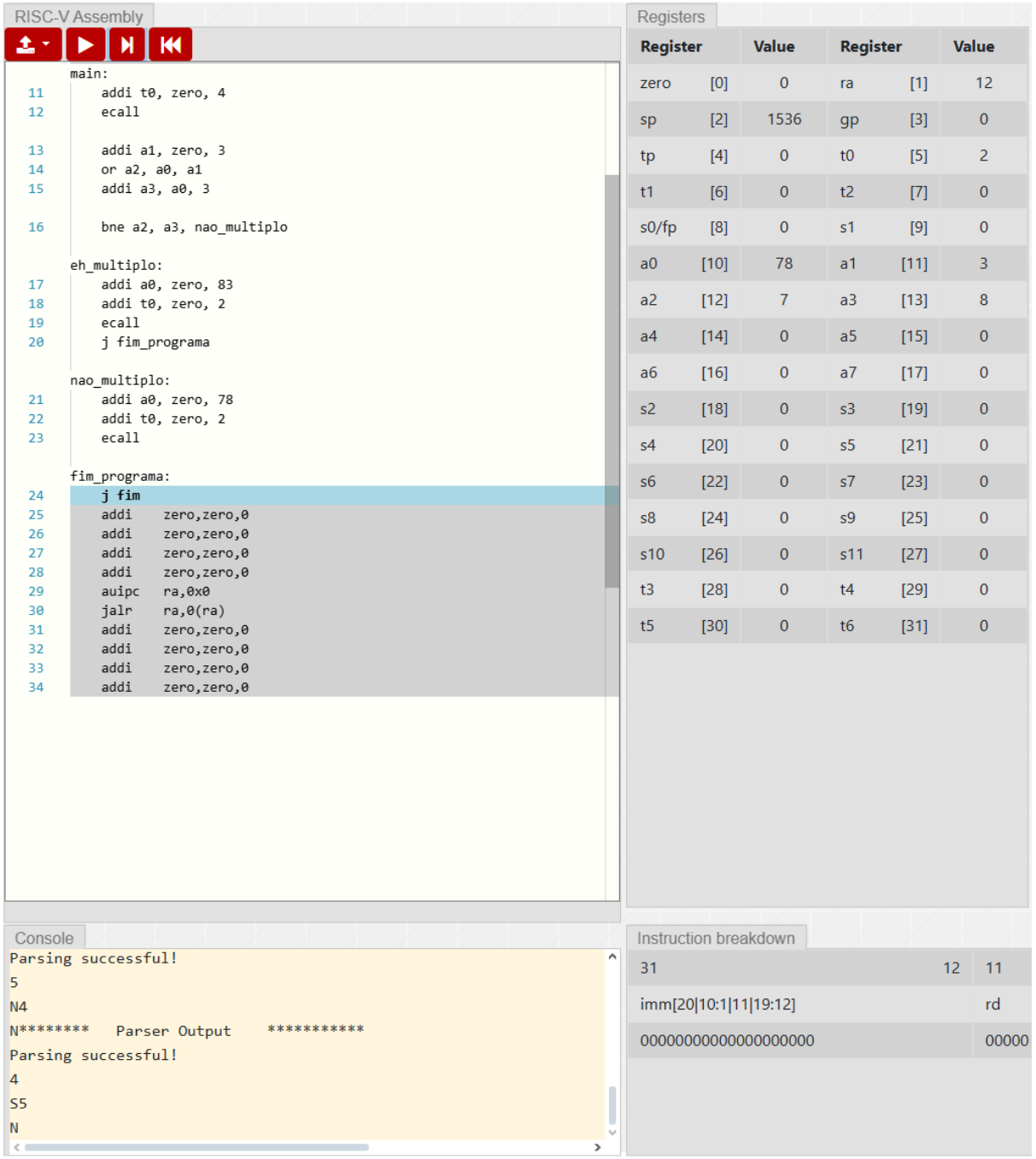
Faça um programa que leia um número do teclado e imprima a letra I se o número for ímpar ou a letra P se o número for par. Busque os códigos da letra I e P na tabela ASCII para poder imprimir o caractere corretamente.



Atividade 3

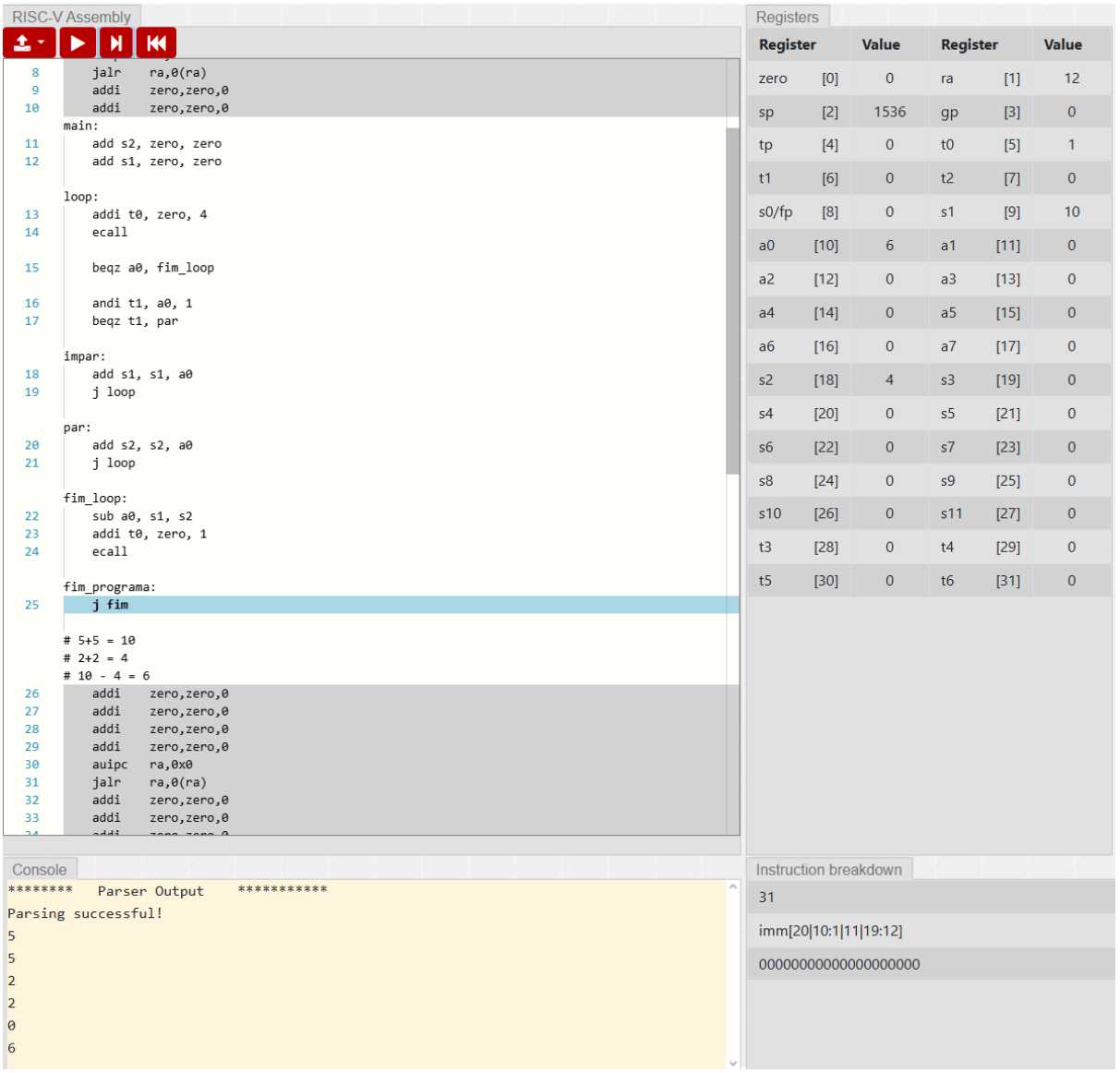
Você consegue fazer outra versão do programa que detecte se um número é múltiplo de 4? Nesse caso, imprima S para sim e N para não.

Dica: A instrução OR faz uma operação OU lógica bit a bit, para todos os bits do número. Então, um OR entre os números 6 (0110) e 5 (0101) tem como resposta o número 7 (0111) pois todos os bits com valor 1 em, ao menos, um dos números foram mantidos como 1 no resultado final.



Atividade 4

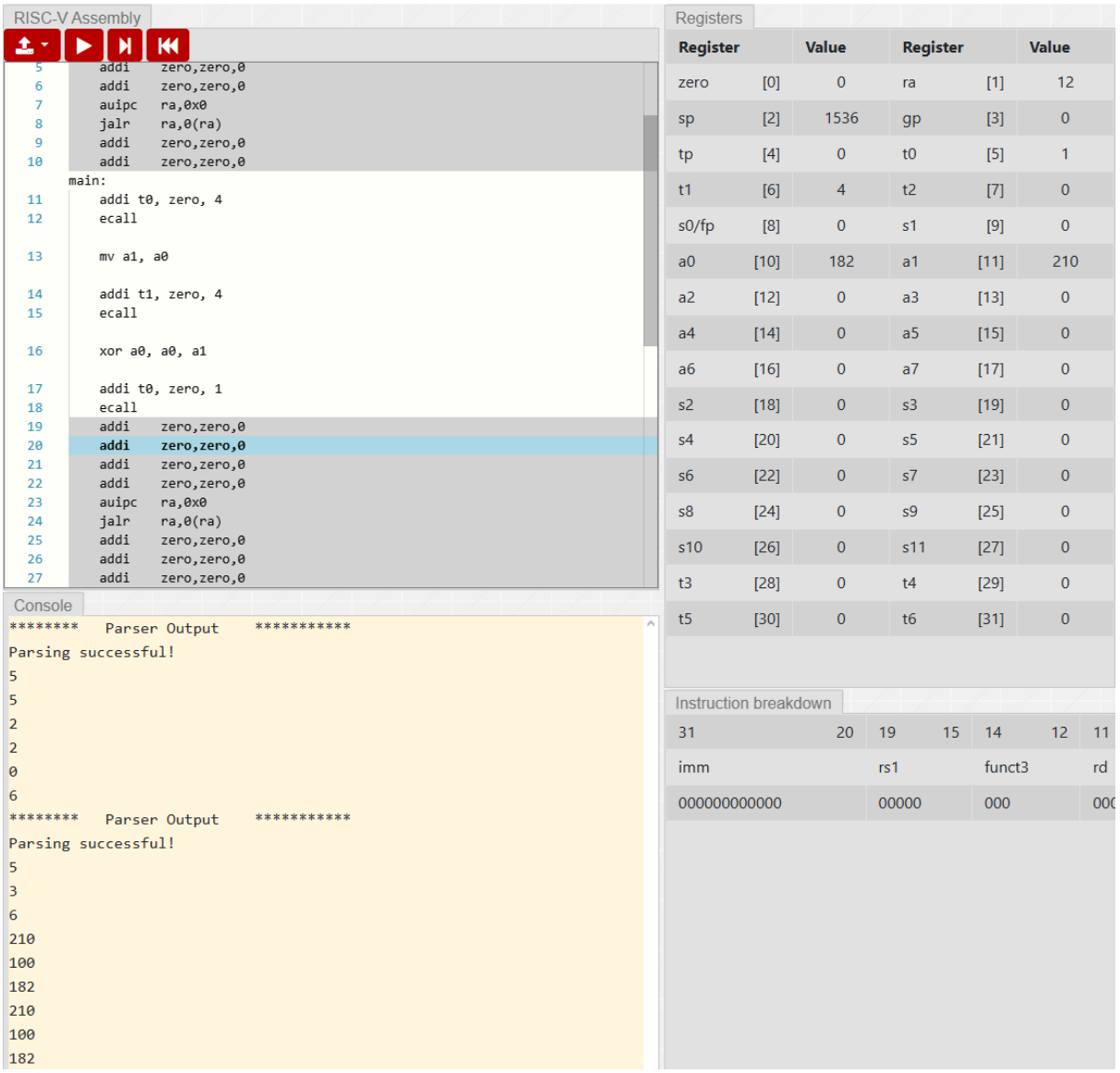
Faça um programa que leia múltiplos números do teclado. Seu programa deve parar quando for digitado o valor 0. Ao final do programa, ele deve imprimir o resultado da soma de todos os ímpares subtraindo da soma de todos os pares. Resultado = Soma(ímpares) - Soma(pares).



Atividade 5

Faça um programa que leia dois números do teclado: o segredo e o número a codificar. O programa deve imprimir o número codificado. Para isso, utilize a instrução XOR. Note que esse programa servirá tanto para codificar quanto para decodificar o número.

⚠️ **Atenção:** Apesar de ser uma forma de codificar um número, essa técnica não é segura. Ela é apenas uma forma de você aprender a utilizar a instrução XOR.



Atividade 6

Faça um programa que leia o segredo do teclado e, depois, leia vários números do teclado imprimindo cada um codificado em sequência. O programa deve parar quando for digitado o valor 0.

RISC-V Assembly

```
0       addi    zero,zero,0
        kernel:
        main:
11          addi t0, zero, 4
12          ecall

13          mv a1, a0

        loop:
14          addi t0, zero, 4
15          ecall

16          beqz a0, fim_programa

17          xor a0, a0, a1

18          addi t0, zero, 1
19          ecall

20          j loop

        fim_programa:
21          addi    zero,zero,0
22          addi    zero,zero,0
23          addi    zero,zero,0
24          addi    zero,zero,0
25          auipc   ra,0x0
26          jalr    ra,0(ra)
27          addi    zero,zero,0
```

Registers

Register		Value	Register		Value
zero	[0]	0	ra	[1]	108
sp	[2]	1536	gp	[3]	0
tp	[4]	0	t0	[5]	4
t1	[6]	0	t2	[7]	0
s0/fp	[8]	0	s1	[9]	0
a0	[10]	0	a1	[11]	10
a2	[12]	0	a3	[13]	0
a4	[14]	0	a5	[15]	0
a6	[16]	0	a7	[17]	0
s2	[18]	0	s3	[19]	0
s4	[20]	0	s5	[21]	0
s6	[22]	0	s7	[23]	0
s8	[24]	0	s9	[25]	0
s10	[26]	0	s11	[27]	0
t3	[28]	0	t4	[29]	0
t5	[30]	0	t6	[31]	0

Console

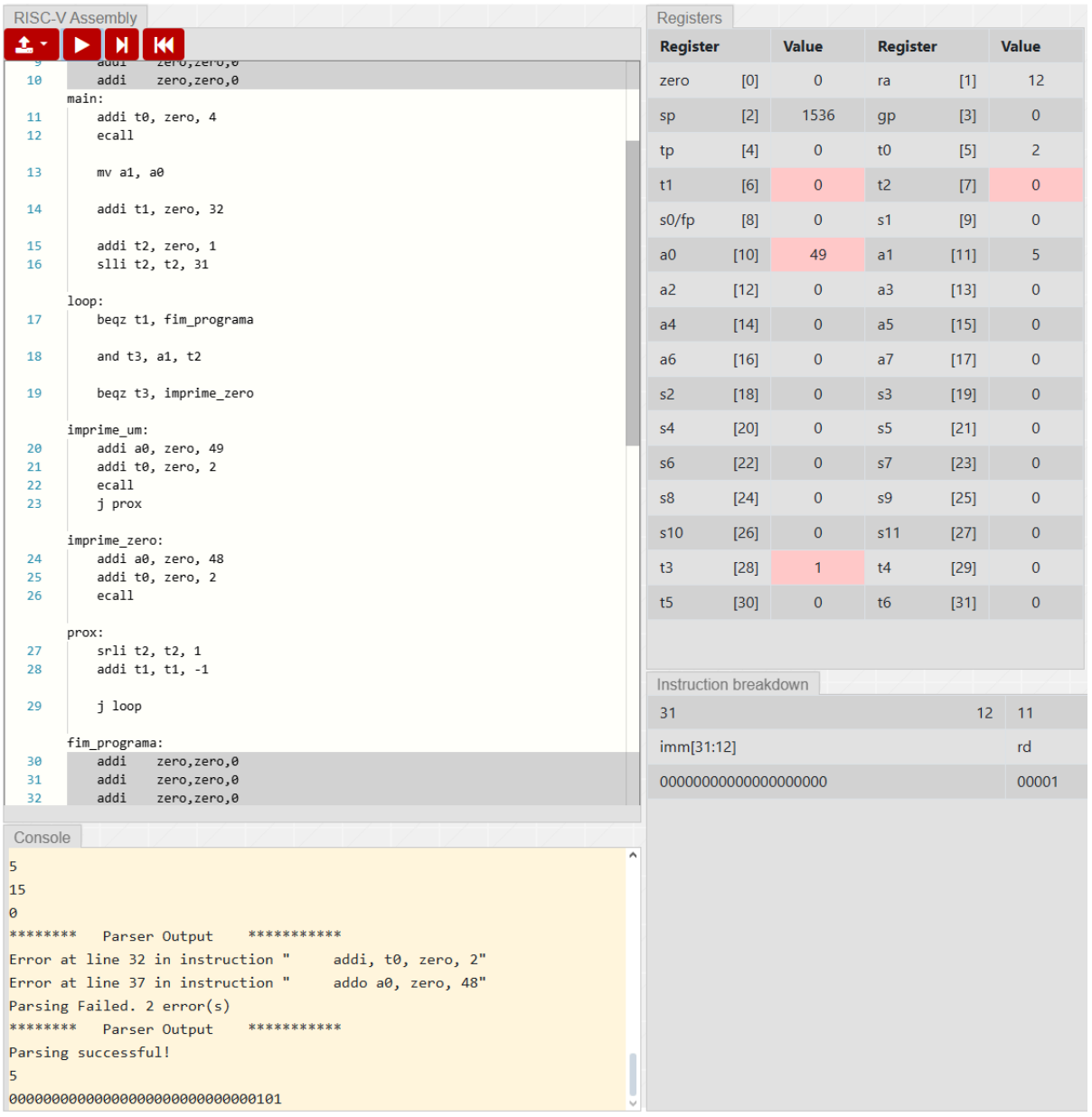
```
20
30
30
********   Parser Output    ***********
Parsing successful!
10
20
30
50
56
11
1
2
8
0
********   Parser Output    ***********
Parsing successful!
10
5
15
0
```

Instruction breakdown

31	12	11
imm[31:12]		rd
00000000000000000000		00001

Atividade 7

Faça um programa que leia um número do teclado e imprima o número em binário. Para isso, utilize as instruções AND e de deslocamento de bits. Como os números do seu processador são de 32 bits, você deve ter um laço for no seu código com 32 interações.



RISC-V Assembly

```
10      addi    zero,zero,0
        main:
11          addi t0, zero, 4
12          ecall

13          mv a1, a0

14          addi t1, zero, 32

15          addi t2, zero, 1
16          slli t2, t2, 31

        loop:
17          beqz t1, fim_programa

18          and t3, a1, t2

19          beqz t3, imprime_zero

        imprime_um:
20          addi a0, zero, 49
21          addi t0, zero, 2
22          ecall
23          j prox

        imprime_zero:
24          addi a0, zero, 48
25          addi t0, zero, 2
26          ecall

        prox:
27          srli t2, t2, 1
28          addi t1, t1, -1

29          j loop

        fim_programa:
30          addi    zero,zero,0
31          addi    zero,zero,0
32          addi    zero,zero,0
```

Registers

Register		Value	Register		Value
zero	[0]	0	ra	[1]	12
sp	[2]	1536	gp	[3]	0
tp	[4]	0	t0	[5]	2
t1	[6]	0	t2	[7]	0
s0/fp	[8]	0	s1	[9]	0
a0	[10]	49	a1	[11]	5
a2	[12]	0	a3	[13]	0
a4	[14]	0	a5	[15]	0
a6	[16]	0	a7	[17]	0
s2	[18]	0	s3	[19]	0
s4	[20]	0	s5	[21]	0
s6	[22]	0	s7	[23]	0
s8	[24]	0	s9	[25]	0
s10	[26]	0	s11	[27]	0
t3	[28]	1	t4	[29]	0
t5	[30]	0	t6	[31]	0

Instruction breakdown

31	12	11
imm[31:12]		rd
00000000000000000000		00001

Console

```
5
15
0
********   Parser Output    ***********
Error at line 32 in instruction "     addi, t0, zero, 2"
Error at line 37 in instruction "     addo a0, zero, 48"
Parsing Failed. 2 error(s)
********   Parser Output    ***********
Parsing successful!
5
00000000000000000000000000000101
```